Online Research @ Cardiff

This is an Open Access document downloaded from ORCA, Cardiff University's institutional repository: https://orca.cardiff.ac.uk/id/eprint/94721/

This is the author's version of a work that was submitted to / accepted for publication.

Citation for final published version:

Cerutti, Federico ORCID: https://orcid.org/0000-0003-0755-0358, Vallati, Mauro and Giacomin, Massimiliano 2017. An efficient java-based solver for abstract argumentation frameworks: jArgSemSAT. International Journal on Artificial Intelligence Tools 26 (2) , 1750002. 10.1142/S0218213017500026 file

Publishers page: http://dx.doi.org/10.1142/S0218213017500026 <http://dx.doi.org/10.1142/S0218213017500026>

Please note:
Changes made as a result of publishing processes such as copy-editing, formatting and page numbers may not be reflected in this version. For the definitive version of this publication, please refer to the published source. You are advised to consult the publisher's version if you wish to cite this paper.

This version is being made available in accordance with publisher policies. See http://orca.cf.ac.uk/policies.html for usage policies. Copyright and moral rights for publications made available in ORCA are retained by the copyright holders.

information services
gwasanaethau gwybodaeth

International Journal on Artificial Intelligence Tools
© World Scientific Publishing Company

An Efficient Java-Based Solver for Abstract Argumentation Frameworks: jArgSemSAT

Federico Cerutti

School of Computing Science & Informatics
Cardiff University,
Queen's Buildings, 5 The Parade
Cardiff, CF24 3AA, United Kingdom
CeruttiF@cardiff.ac.uk

Mauro Vallati

School of Computing and Engineering
University of Huddersfield, Queensgate
Huddersfield, HD1 3DH, United Kingdom
m.vallati@hud.ac.uk

Massimiliano Giacomin

Dipartimento di Ingegneria dell'Informazione
Università degli Studi di Brescia
Brescia, 25123, Italy
massimiliano.giacomin@unibs.it

Received (Day Month Year)
Revised (Day Month Year)
Accepted (Day Month Year)

Dung's argumentation frameworks are adopted in a variety of applications, from argument-mining, to intelligence analysis and legal reasoning. Despite this broad spectrum of already existing applications, the mostly adopted solver—in virtue of its simplicity—is far from being comparable to the current state-of-the-art solvers. On the other hand, most of the current state-of-the-art solvers are far too complicated to be deployed in real-world settings. In this paper we provide and extensive description of jArgSemSAT, a Java re-implementation of ArgSemSAT. ArgSemSAT represents the best single solver for argumentation semantics with the highest level of computational complexity. We show that jArgSemSAT can be easily integrated in existing argumentation systems (1) as an off-the-shelf, standalone, library; (2) as a Tweety compatible library; and (3) as a fast and robust web service freely available on the Web. Our large experimental analysis shows that—despite being written in Java—jArgSemSAT would have scored in most of the cases among the three bests solvers for the two semantics with highest computational complexity—Stable and Preferred—in the last competition on computational models of argumentation.

Keywords: Abstract Argumentation; Argumentation Semantics; Off-The-Shelf Solver.

1. Introduction

Dung's theory of abstract argumentation[1] is a unifying framework able to encompass a large variety of specific formalisms in the areas of nonmonotonic reasoning, logic programming and computational argumentation. It is based on the notion of argumentation framework (AF), that consists of a set of arguments and an *attack* relation between them. Different *argumentation semantics* introduce in a declarative way the criteria to determine which arguments emerge as "justified" from the conflict, by identifying a number of *extensions*, i.e. sets of arguments that can "survive the conflict together". In Dung's paper[1] three "traditional" semantics are introduced, namely *grounded*, *stable*, and *preferred* semantics, as well as the auxiliary notion of *complete* extension, to highlight the linkage between grounded and preferred semantics. Other literature proposals include *semi-stable*[2], *ideal*[3], and *CF2*[4] semantics.

The preferred semantics represents one of the main contributions in Dung's theory[1] and is widely adopted—among other areas—in decision support systems[5] and in critical thinking support systems[6], as it allows multiple extensions (differently from grounded semantics), the existence of extensions is always guaranteed (differently from stable semantics), and no extension is a proper subset of another extension. The investigation on alternative argumentation semantics is an active research area since two decades[7].

Many problems associated to preferred, but also to stable, semantics turn to be at the high levels of the polynomial hierarchy[8]. In this paper we will focus on four problems, namely *credulous* and *skeptical* acceptance of an argument with respect to a given argumentation framework and a given semantics and *enumeration* of *all* or *some* semantics extensions given an argumentation framework. Those are the problems considered in the first *International Competition on Computational Models of Argumentation* (ICCMA2015) that determined the state-of-the-art of the current implementations for addressing the above problems with respect to the three aforementioned semantics (plus the complete extensions).

Surprisingly, the winner of ICCMA2015—CoQuiAAS[a] [9]—never scored in the first two positions with respect to the most computationally expensive semantics, namely stable and preferred semantics. Indeed, CoQuiAAS performed very well on grounded semantics—where each problem is polynomial—thanks to a very efficient unit propagation mechanism, as well as on the tracks associated to complete extensions problems. The interested reader is referred to the competition summary[10] and to the competition website[b] for an overview of the results.

Instead, ArgSemSAT is the best single solver when facing semantics with an high level of computational complexity. It is constantly either first or second placed in each track associated to stable and preferred semantics—except one due to an im-

[a] `http://www.cril.univ-artois.fr/coquiaas/`
[b] `http://argumentationcompetition.org/2015/`

plementation bug discovered after the competition[c]. Despite this bug, ArgSemSAT scored second—at one single Borda count point from CoQuiAAS—and the overhead in solving problems associated to grounded semantics is—on average—of 3.88 seconds from CoQuiASS. Therefore, this difference is neglectable in most real-world situation, when ultimately a human user will consume the result of argumentation-based reasoning procedures.

Building on top of the success of ArgSemSAT, we introduced jArgSemSAT that is specifically designed for being easily integrated within existing argumentation systems. Indeed, ArgSemSAT—as well as many other solvers that participated in ICCMA2015—is written in C++ and requires an external SAT solver as an NP oracle. This can hardly be considered an off-the-shelf system, as most of the current tools using argumentation technology are based on existing Java approaches (such as Dung-O-Matic[11], adopted e.g. in CISpaces[6]), or on the Tweety libraries for knowledge representation and reasoning[12], or use a web-service interface as ArgTech[13]. We developed jArgSemSAT in Java, with a specific focus on being compatible with Dung-O-Matic, Tweety, and with a web-service interface in turn compatible with ArgTech. A large experimental analysis confirms that jArgSemSAT—despite being written in Java—would have been one of the best solvers for most of the ICCMA2015 tracks associated to the two semantics with highest computational complexity. Therefore, not only ArgSemSAT is compatible with existing technology, but it is also among the best solvers for stable and preferred semantics.

This paper is an extension of the short *report from the field* work presented at the 15th Conference on Principles of Knowledge Representation and Reasoning[14] with substantial additional material, including: (i) a complete description of jArgSemSAT which includes a thorough description of the implemented algorithms and processes; (ii) a significantly extended experimental analysis, that considers a comparison with the state of the art—including solvers which took part in ICCMA2015—on several problems associated to stable and preferred semantics; and (iii) an extensive discussion of the benefits of employing jArgSemSAT within CISpaces[6].

The rest of the paper is organised as follows. Section 2 provides the required background on abstract argumentation; Section 3 gives an overview of the jArgSemSAT system, while Section 4 focuses on system design; Section 5 reports our experimental results; finally, conclusions and discussions of the benefits of jArgSemSAT are given in Section 6.

[c]Details can be found in `http://downloads.sourceforge.net/project/argsemsat/ArgSemSAT-1.0rc3/ArgSemSAT_1.0rc3.zip`

2. Background

2.1. *Argumentation frameworks and semantics*

An argumentation framework[1] consists of a set of arguments[d] and a binary attack relation between them.

Definition 2.1. An *argumentation framework* (AF) is a pair $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$ where $\mathcal{A}$ is a set of arguments and $\mathcal{R} \subseteq \mathcal{A} \times \mathcal{A}$. We say that $\mathbf{b}$ *attacks* $\mathbf{a}$ iff $\langle \mathbf{b}, \mathbf{a} \rangle \in \mathcal{R}$, also denoted as $\mathbf{b} \rightarrow \mathbf{a}$. The set of attackers of an argument $\mathbf{a}$ will be denoted as $\mathbf{a}^- \triangleq \{\mathbf{b} : \mathbf{b} \rightarrow \mathbf{a}\}$, the set of arguments attacked by $\mathbf{a}$ will be denoted as $\mathbf{a}^+ \triangleq \{\mathbf{b} : \mathbf{a} \rightarrow \mathbf{b}\}$. We also extend these notations to sets of arguments, i.e. given $E \subseteq \mathcal{A}$, $E^- \triangleq \{\mathbf{b} \mid \exists \mathbf{a} \in E, \mathbf{b} \rightarrow \mathbf{a}\}$ and $E^+ \triangleq \{\mathbf{b} \mid \exists \mathbf{a} \in E, \mathbf{a} \rightarrow \mathbf{b}\}$.

An argument $\mathbf{a}$ without attackers, i.e. such that $\mathbf{a}^- = \emptyset$, is said *initial*. Moreover, each argumentation framework has an associated directed graph where the vertices represent the arguments, and the edges represent the attacks.

The basic properties of conflict–freeness, acceptability, and admissibility of a set of arguments are fundamental for the definition of argumentation semantics.

Definition 2.2. Given an AF $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$:

- a set $S \subseteq \mathcal{A}$ is a *conflict–free* set of Γ if $\nexists\, \mathbf{a}, \mathbf{b} \in S$ s.t. $\mathbf{a} \rightarrow \mathbf{b}$;
- an argument $\mathbf{a} \in \mathcal{A}$ is *acceptable* with respect to a set $S \subseteq \mathcal{A}$ of Γ if $\forall \mathbf{b} \in \mathcal{A}$ s.t. $\mathbf{b} \rightarrow \mathbf{a}$, $\exists\, \mathbf{c} \in S$ s.t. $\mathbf{c} \rightarrow \mathbf{b}$;
- the function $\mathrm{F}_\Gamma : 2^{\mathcal{A}} \rightarrow 2^{\mathcal{A}}$ such that $\mathrm{F}_\Gamma(S) = \{\mathbf{a} \mid \mathbf{a} \text{ is acceptable w.r.t. } S\}$ is called the *characteristic function* of Γ;
- a set $S \subseteq \mathcal{A}$ is an *admissible* set of Γ if S is a conflict–free set of Γ and every element of S is acceptable with respect to S, i.e. $S \subseteq \mathrm{F}_\Gamma(S)$;
- a set $S \subseteq \mathcal{A}$ is a *complete* extension of Γ if S is an admissible set of Γ such that it contains each argument acceptable with respect to S, i.e. $S = \mathrm{F}_\Gamma(S)$.

An argumentation semantics σ prescribes for any AF Γ a set of *extensions*, denoted as $\mathcal{E}_\sigma(\Gamma)$, namely a set of sets of arguments satisfying the conditions dictated by σ. Here we need to recall the definitions of grounded (denoted as GR), stable (denoted as ST), and preferred (denoted as PR) semantics only.

Definition 2.3. Given an AF $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$:

- a set $S \subseteq \mathcal{A}$ is the *grounded extension* of Γ, i.e. $S \in \mathcal{E}_{\mathsf{GR}}(\Gamma)$, iff S is the least fixed point of F_Γ;
- a set $S \subseteq \mathcal{A}$ is a *stable extension* of Γ, i.e. $S \in \mathcal{E}_{\mathsf{ST}}(\Gamma)$, iff S is a conflict-free set of Γ and $S \cup S^+ = \mathcal{A}$;

[d]In this paper we consider only *finite* sets of arguments: see Ref. 15 for a discussion on infinite sets of arguments.

- a set $S \subseteq \mathcal{A}$ is a *preferred extension* of Γ, i.e. $S \in \mathcal{E}_{\mathsf{PR}}(\Gamma)$, iff S is a maximal (w.r.t. set inclusion) admissible set of Γ.

The notion of complete extension has been introduced[1] as a linkage between preferred and grounded semantics. Given an *AF* $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$, a set $S \subseteq \mathcal{A}$ is a *complete extension* of Γ iff S is a conflict-free set of Γ and $S = \mathrm{F}_\Gamma(S)$. The auxiliary notion of complete extension provides a mean for re-defining the grounded extension as the minimal (with respect to set inclusion) complete extension, and a preferred extension as a maximal (w.r.t. set inclusion) complete extension.

Each extension S implicitly defines a three-valued *labelling* of arguments: an argument $\mathbf{a}$ is labelled `in` iff $\mathbf{a} \in S$; is labelled `out` iff $\exists\ \mathbf{b} \in S$ s.t. $\mathbf{b} \rightarrow \mathbf{a}$; is labelled `undec` if neither of the above conditions holds. In the light of this correspondence, argumentation semantics can be equivalently defined in terms of labellings rather than of extensions[16,7].

Definition 2.4. Given a set of arguments S, a *labelling* of S is a total function $\mathcal{L}ab : S \longrightarrow \{\mathtt{in}, \mathtt{out}, \mathtt{undec}\}$. The set of all *labellings* of S is denoted as $\mathfrak{L}_S$. Given an *AF* $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$, a *labelling* of Γ is a labelling of $\mathcal{A}$. The set of all *labellings* of Γ is denoted as $\mathfrak{L}(\Gamma)$.

In particular, complete labellings can be defined as follows.

Definition 2.5. Let $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$ be an argumentation framework. A labelling $\mathcal{L}ab \in \mathfrak{L}(\Gamma)$ is a *complete labelling* of Γ iff it satisfies the following conditions for any $\mathbf{a} \in \mathcal{A}$:

- $\mathcal{L}ab(\mathbf{a}) = \mathtt{in} \Leftrightarrow \forall \mathbf{b} \in \mathbf{a}^- \mathcal{L}ab(\mathbf{b}) = \mathtt{out}$;
- $\mathcal{L}ab(\mathbf{a}) = \mathtt{out} \Leftrightarrow \exists \mathbf{b} \in \mathbf{a}^- : \mathcal{L}ab(\mathbf{b}) = \mathtt{in}$;

The grounded, stable, and preferred labelling can then be defined on the basis of complete labellings.

Definition 2.6. Let $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$ be an argumentation framework. A labelling $\mathcal{L}ab \in \mathfrak{L}(\Gamma)$ is

- the *grounded labelling* of Γ if it is the complete labelling of Γ maximising the set of arguments labelled `undec`;
- a *stable labelling* Γ if it is a complete labelling of Γ and there is no argument labelled `undec`;
- a *preferred labelling* of Γ if it is a complete labelling of Γ maximizing the set of arguments labelled `in`.

In order to show the connection between extensions and labellings, let us recall[7] the definition of the function `Ext2Lab`, returning the labelling corresponding to a conflict–free set of arguments S.

Definition 2.7. Given an *AF* $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$ and a conflict–free set S of Γ, the corresponding labelling $\mathtt{Ext2Lab}(S)$ is the labelling of Γ $\mathcal{L}ab$ where

- $\mathcal{L}ab(\mathbf{a}) = \texttt{in} \Leftrightarrow \mathbf{a} \in S$
- $\mathcal{L}ab(\mathbf{a}) = \texttt{out} \Leftrightarrow \exists\ \mathbf{b} \in S$ s.t. $\mathbf{b} \rightarrow \mathbf{a}$
- $\mathcal{L}ab(\mathbf{a}) = \texttt{undec} \Leftrightarrow \mathbf{a} \notin S \wedge \nexists\ \mathbf{b} \in S$ s.t. $\mathbf{b} \rightarrow \mathbf{a}$

There is a bijective correspondence between the complete, grounded, stable, preferred extensions and the complete, grounded, stable, preferred labellings, respectively.[16]

Proposition 2.1. *Given an an AF* $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$*,* $\mathcal{L}ab$ *is a complete (grounded, stable, preferred) labelling of* Γ *if and only if there is a complete (grounded, stable, preferred) extension* S *of* Γ *such that* $\mathcal{L}ab = \texttt{Ext2Lab}(S)$*.*

2.2. *Computational Problems in Abstract Argumentation*

Table 1. Computational complexity of credulous and skeptical acceptance in finite AFs w.r.t. the three semantics introduced by Dung.

	Semantics σ		
	GR	ST	PR
DC-σ	in P	NP-compl.	NP-compl.
DS-σ	in P	coNP-compl.	Π_2^p-compl.

Credulous and skeptical acceptance of an argument are the two most studied decision problems in argumentation theory (see Ref. 8).

An argument $\mathbf{a}$ is *credulously* accepted with respect to a given semantics σ and a given AF Γ iff $\mathbf{a}$ belongs to at least one extension of Γ under σ: $\exists E \in \mathcal{E}_\sigma(\Gamma)$ s.t. $\mathbf{a} \in E$. We denote such a problem as DC-σ. An argument $\mathbf{a}$ is *skeptically* accepted with respect to a given semantics σ and a given AF Γ iff $\mathbf{a}$ belongs to each extension of Γ under σ: $\forall E \in \mathcal{E}_\sigma(\Gamma)$ $\mathbf{a} \in E$. We denote this problem as DS-σ.

The complexity of DC-σ and DS-σ when σ is the stable or preferred semantics lies at the first or second level of the polynomial hierarchy, as shown in Table 1 (see Ref. 8).[e]

In addition to credulous and skeptical acceptance, the following two problems are worth considering and have been included in ICCMA2015:

- given an AF, determine some extension (SE) of a given semantics;
- given an AF, determine all extensions (EE) of a given semantics.

[e]Computational complexity for credulous and skeptical acceptance w.r.t. admissible sets, as well as w.r.t. complete extensions can easily been identified, cf. Ref. 8. In particular, credulous acceptance w.r.t. admissible and complete extensions is equivalent to credulous acceptance w.r.t. preferred semantics, skeptical acceptance w.r.t. admissible sets is trivial, and skeptical acceptance w.r.t. complete extensions is equivalent to acceptance w.r.t. grounded semantics.

Table 2. First three places in ICCMA2015 with respect to stable and preferred semantics.

	Semantics σ	
	ST	PR
DC-σ	1.ASPARTIX-D	1.ArgSemSAT
	2.ArgSemSAT	2.LabSATSolver
	3. LabSATSolver	3. CoQuiAAS
DS-σ	1. ASPARTIX-D	1. ArgSemSAT
	2. LabSATSolver	2. Cegartix
	3. CoQuiAAS	3. LabSATSolver
SE-σ	1. ASPARTIX-D	1. Cegartix
	2. ArgSemSAT	2. ArgSemSAT
	3. LabSATSolver	3. LabSATSolver
EE-σ	1. ASPARTIX-D	1. Cegartix
	2. ArgSemSAT	2. ArgSemSAT
	3. CoQuiAAS	3. CoQuiAAS

As shown in Table 2, ArgSemSAT is always in the first two positions both in the case of stable and preferred semantics, except for DS-ST due to an implementation bug discovered after the competition.

ArgSemSAT scored second considering the Borda count across all tracks of ICCMA2015[f], at one point of distance from CoQuiAAS, which scored at most third in the tracks associated to stable and preferred semantics, but was constantly the best for grounded semantics. For this semantics, CoQuiAAS uses an efficient unit propagation mechanism, while ArgSemSAT searches for it via a maximisation process in the space of complete labellings. Despite this massive difference in the approaches, the difference of execution times between CoQuiAAS and ArgSemSAT over the competition benchmark and with respect to the four tracks related to grounded semantics is of 3.88 seconds on average (standard deviation 5.89).

3. Overview of jArgSemSAT

jArgSemSAT and ArgSemSAT enumerate preferred extensions by multiple calls to a SAT solver[17]. A propositional formula over a set of Boolean variables is satisfiable iff there exists a truth assignment of the variables such that the formula evaluates to True. Checking whether such an assignment exists is the satisfiability (SAT) problem. jArgSemSAT and ArgSemSAT exploit an encoding of complete extensions as a propositional formula in Conjunctive Normal Form (CNF) and apply a filtering

[f] `http://argumentationcompetition.org/2015/results.html`

procedure over the space of complete extensions to select the maximal ones, i.e. the preferred extensions.

There are therefore three main components within jArgSemSAT and ArgSemSAT: (1) a propositional formula for complete labelling—cf. Definition 2.5; (2) an NP-oracle in the form of a SAT solver; and (3) a filtering process.

3.1. *Propositional formulae for complete labellings*

Given an AF $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$ we identify a propositional formula Π_Γ such that each satisfying assignment of the formula corresponds to a complete labelling. Such formula is based on Definition 2.5, which however admits several logically equivalent propositional encodings that lead to severely different performance[17].

As a first step to explore alternative encodings, the conditions in Definition 2.5 can be redundantly expressed as a conjunction of 6 terms, i.e. $C^{\rightarrow}_{\mathtt{in}} \wedge C^{\leftarrow}_{\mathtt{in}} \wedge C^{\rightarrow}_{\mathtt{out}} \wedge C^{\leftarrow}_{\mathtt{out}} \wedge C^{\rightarrow}_{\mathtt{undec}} \wedge C^{\leftarrow}_{\mathtt{undec}}$, where

- $C^{\rightarrow}_{\mathtt{in}} \equiv (\mathcal{L}ab(\mathbf{a}) = \mathtt{in} \Rightarrow \forall \mathbf{b} \in \mathbf{a}^- \mathcal{L}ab(\mathbf{b}) = \mathtt{out})$;
- $C^{\leftarrow}_{\mathtt{in}} \equiv (\mathcal{L}ab(\mathbf{a}) = \mathtt{in} \Leftarrow \forall \mathbf{b} \in \mathbf{a}^- \mathcal{L}ab(\mathbf{b}) = \mathtt{out})$;
- $C^{\rightarrow}_{\mathtt{out}} \equiv (\mathcal{L}ab(\mathbf{a}) = \mathtt{out} \Rightarrow \exists \mathbf{b} \in \mathbf{a}^- : \mathcal{L}ab(\mathbf{b}) = \mathtt{in})$;
- $C^{\leftarrow}_{\mathtt{out}} \equiv (\mathcal{L}ab(\mathbf{a}) = \mathtt{out} \Leftarrow \exists \mathbf{b} \in \mathbf{a}^- : \mathcal{L}ab(\mathbf{b}) = \mathtt{in})$;
- $C^{\rightarrow}_{\mathtt{undec}} \equiv (\mathcal{L}ab(\mathbf{a}) = \mathtt{undec} \Rightarrow \forall \mathbf{b} \in \mathbf{a}^- \mathcal{L}ab(\mathbf{b}) \neq \mathtt{in} \wedge \exists \mathbf{c} \in \mathbf{a}^- : \mathcal{L}ab(\mathbf{c}) = \mathtt{undec})$;
- $C^{\leftarrow}_{\mathtt{undec}} \equiv (\mathcal{L}ab(\mathbf{a}) = \mathtt{undec} \Leftarrow \forall \mathbf{b} \in \mathbf{a}^- \mathcal{L}ab(\mathbf{b}) \neq \mathtt{in} \wedge \exists \mathbf{c} \in \mathbf{a}^- : \mathcal{L}ab(\mathbf{c}) = \mathtt{undec})$.

Moreover we define $C^{\leftrightarrow}_{\mathtt{in}} \equiv C^{\rightarrow}_{\mathtt{in}} \wedge C^{\leftarrow}_{\mathtt{in}}$, $C^{\leftrightarrow}_{\mathtt{out}} \equiv C^{\rightarrow}_{\mathtt{out}} \wedge C^{\leftarrow}_{\mathtt{out}}$, $C^{\leftrightarrow}_{\mathtt{undec}} \equiv C^{\rightarrow}_{\mathtt{undec}} \wedge C^{\leftarrow}_{\mathtt{undec}}$.

We identify[17] 5 non redundant strict subsets of the above six terms that equivalently characterize complete extensions[g], namely: (i) $C^{\leftrightarrow}_{\mathtt{in}} \wedge C^{\leftrightarrow}_{\mathtt{out}}$, (ii) $C^{\leftrightarrow}_{\mathtt{out}} \wedge C^{\leftrightarrow}_{\mathtt{undec}}$, (iii) $C^{\leftrightarrow}_{\mathtt{in}} \wedge C^{\leftrightarrow}_{\mathtt{undec}}$, (iv) $C^{\rightarrow}_{\mathtt{in}} \wedge C^{\rightarrow}_{\mathtt{out}} \wedge C^{\rightarrow}_{\mathtt{undec}}$, (v) $C^{\leftarrow}_{\mathtt{in}} \wedge C^{\leftarrow}_{\mathtt{out}} \wedge C^{\leftarrow}_{\mathtt{undec}}$.

SAT solvers require such constraints in conjunctive normal form (CNF). Letting $k = |\mathcal{A}|$ we define a bijection $\phi : \{1, \ldots, k\} \mapsto \mathcal{A}$ (the inverse map is denoted as ϕ^{-1}). ϕ is an indexing of $\mathcal{A}$: for sake of brevity we might refer to the argument $\phi(i)$ as "argument i." For each argument i we define three Boolean variables, I_i, O_i, and U_i, with the intended meaning that I_i (resp O_i, U_i) is True when argument i is labelled `in` (resp. `out`, `undec`), False otherwise. Given $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$ we define the corresponding set of variables as $\mathcal{V}(\Gamma) \triangleq \cup_{1 \leq i \leq |\mathcal{A}|} \{I_i, O_i, U_i\}$.

The conjunction of the following formulae in CNF format is equivalent to $C^{\leftrightarrow}_{\mathtt{in}} \wedge C^{\leftrightarrow}_{\mathtt{out}} \wedge C^{\leftrightarrow}_{\mathtt{undec}}$:

[g] $C^{\leftrightarrow}_{\mathtt{in}} \wedge C^{\leftrightarrow}_{\mathtt{out}}$ and $C^{\rightarrow}_{\mathtt{in}} \wedge C^{\rightarrow}_{\mathtt{out}} \wedge C^{\rightarrow}_{\mathtt{undec}}$ correspond to the alternative definitions of complete labellings in Ref. 18, where a proof of their equivalence is provided.

$$\bigwedge_{i\in\{1,\ldots,k\}} (I_i \vee O_i \vee U_i) \wedge (\neg I_i \vee \neg O_i) \wedge(\neg I_i \vee \neg U_i) \wedge (\neg O_i \vee \neg U_i) \tag{1}$$

$$\bigwedge_{\{i|\phi(i)^-=\emptyset\}} I_i \wedge \neg O_i \wedge \neg U_i \tag{2}$$

$$\bigwedge_{\{i|\phi(i)^-\neq\emptyset\}} \bigwedge_{\{j|\phi(j)\to\phi(i)\}} \neg I_i \vee O_j \tag{3}$$

$$\bigwedge_{\{i|\phi(i)^-\neq\emptyset\}} I_i \vee \bigvee_{\{j|\phi(j)\to\phi(i)\}} (\neg O_j) \tag{4}$$

$$\bigwedge_{\{i|\phi(i)^-\neq\emptyset\}} \neg O_i \vee \bigvee_{\{j|\phi(j)\to\phi(i)\}} I_j \tag{5}$$

$$\bigwedge_{\{i|\phi(i)^-\neq\emptyset\}} \bigwedge_{\{j|\phi(j)\to\phi(i)\}} \neg I_j \vee O_i \tag{6}$$

$$\bigwedge_{\{i|\phi(i)^-\neq\emptyset\}} \bigwedge_{\{j|\phi(j)\to\phi(i)\}} \neg U_i \vee \neg I_j \wedge \neg U_i \vee \bigvee_{\{j|\phi(j)\to\phi(i)\}} U_j \tag{7}$$

$$\bigwedge_{\{i|\phi(i)^-\neq\emptyset\}} \bigwedge_{\{k|\phi(k)\to\phi(i)\}} U_i \vee \neg U_k \vee \bigvee_{\{j|\phi(j)\to\phi(i)\}} I_j \tag{8}$$

where (1) $\wedge$ (3) $\equiv C^{\rightarrow}_{\mathtt{in}}$; (1) $\wedge$ (2) $\wedge$ (4) $\equiv C^{\leftarrow}_{\mathtt{in}}$; (1) $\wedge$ (5) $\equiv C^{\rightarrow}_{\mathtt{out}}$; (1) $\wedge$ (6) $\equiv C^{\leftarrow}_{\mathtt{out}}$; (1) $\wedge$ (7) $\equiv C^{\rightarrow}_{\mathtt{undec}}$; (1) $\wedge$ (8) $\equiv C^{\leftarrow}_{\mathtt{undec}}$.

Users can choose the desired encoding of complete labellings, i.e. the formula Π_Γ, by specifying a sequence of 6 Boolean values—0 for $\bot$ and 1 for $\top$, corresponding to the sequence $\langle C^{\rightarrow}_{\mathtt{in}}, C^{\leftarrow}_{\mathtt{in}}, C^{\rightarrow}_{\mathtt{out}}, C^{\leftarrow}_{\mathtt{out}}, C^{\rightarrow}_{\mathtt{undec}}, C^{\leftarrow}_{\mathtt{undec}}\rangle$. For instance, the sequence 101010 identifies $C^{\rightarrow}_{\mathtt{in}} \wedge C^{\rightarrow}_{\mathtt{out}} \wedge C^{\rightarrow}_{\mathtt{undec}}$. Incorrect configurations that do not correspond to encodings of complete labellings—e.g. 000000—are discarded and the user receives an error message.

3.2. *SAT solvers*

The second component of jArgSemSAT and ArgSemSAT is an NP-oracle in the form of a SAT solver, whose efficiency can be very sensible to the chosen encoding[17]. Following the experience of ArgSemSAT, jArgSemSAT allows the user to choose any desired SAT solver—whose full path must be provided—that supports the DIMACS format, accepts a CNF from the STDIN, and returns a model to the STDOUT. For instance, jArgSemSAT can—as ArgSemSAT does—use GLUCOSE[19] as an external SAT solver.

In order to provide an off-the-shelf solver, jArgSemSAT also integrates as a library Sat4j[20]. Sat4j[h] is an open source library which allows Java programmers to access cross-platform SAT-based solvers. The Sat4j library project started in 2004 as an implementation in Java of the MiniSAT specification[21]. It has been developed with the spirit to keep the technology easily accessible to a newcomer. For instance, it allows the Java programmer to express constraints on objects and hides all the mapping to the various research community input formats from the user.

By default, jArgSemSAT utilises Sat4j with the encoding 111100—equivalent to $C^{\leftrightarrow}_{\texttt{in}} \wedge C^{\leftrightarrow}_{\texttt{out}}$—that on average performs best on MiniSAT-based approaches[17].

3.3. *Filtering process*

Computing grounded, stable and preferred labellings is then a question of implementing efficient filters of complete labellings (see Definition 2.6) that can be computed as a SAT assignment of a propositional formula Π_Γ variables.

Stable labellings—i.e. complete labellings with no `undec` arguments—are the solutions to the formula $\Pi'_\Gamma := \Pi_\Gamma \wedge \bigwedge_{\mathbf{a}\in\mathcal{A}} \neg U_{\phi^{-1}(\mathbf{a})}$. Each time the SAT solver finds a solution *sol*, the formula Π'_Γ is updated to $\Pi'_\Gamma \wedge \neg sol$ and the SAT solver is called on Π'_Γ in order to find an additional stable labelling. The process is iterated until the SAT solver returns no solution, thus enumerating all stable labellings.

Preferred extensions are computed as per Algorithm 1, that is an—unpublished—evolution of the algorithm presented in previous work[17].

Algorithm 1 consists of two nested loops. The external one—lines 5–22—iterates over a (sub)set of complete labellings to identify preferred labellings, while the internal one—lines 8–17—performs an optimisation procedure on a complete labelling to maximise the set of `in`-labelled arguments. Algorithm 1 uses four auxiliary functions. *SatS* refers to a SAT solver able to prove unsatisfiability too: it accepts as input a CNF formula and returns a variable assignment satisfying the formula if it exists, ε otherwise. I-ARGS (resp. O-ARGS, U-ARGS) accepts as input a variable assignment concerning $\mathcal{V}(\Gamma)$ and returns the corresponding set of arguments labelled as `in` (resp. `out`, `undec`).

There are two variables that play a pivotal role in Algorithm 1: *cnf* and *cnfdf*. The former, *cnf*, keeps track of the complete labellings already visited, and thus af-

[h] `http://www.sat4j.org/`

Algorithm 1 Enumeration of Preferred Extensions

1: **Input:** $\Gamma = \langle \mathcal{A}, \mathcal{R} \rangle$
2: **Output:** $E_p \subseteq 2^{\mathcal{A}}$
3: $E_p := \emptyset$
4: $cnf := \Pi_\Gamma \wedge \bigvee_{\mathbf{a} \in \mathcal{A}} I_{\phi^{-1}(\mathbf{a})}$
5: **repeat**
6: $\quad cnfdf := cnf$
7: $\quad prefcand := \emptyset$
8: $\quad$ **repeat**
9: $\quad\quad aCompl := SatS(cnfdf)$
10: $\quad\quad$ **if** $aCompl \neq \varepsilon$ **then**
11: $\quad\quad\quad prefcand := aCompl$
12: $\quad\quad\quad$ **if** U-ARGS($aCompl$) $\neq \emptyset$ **then**
13: $$cnfdf := cnfdf \wedge \bigwedge_{\mathbf{a} \in \text{I-ARGS}(aCompl)} I_{\phi^{-1}(\mathbf{a})} \wedge \bigwedge_{\mathbf{a} \in \text{O-ARGS}(aCompl)} O_{\phi^{-1}(\mathbf{a})} \wedge \bigvee_{\mathbf{a} \in \text{U-ARGS}(aCompl)} I_{\phi^{-1}(\mathbf{a})}$$
14: $\quad\quad\quad$ **end if**
15: $$cnf := cnf \wedge \bigvee_{\mathbf{a} \in \mathcal{A} \setminus \text{I-ARGS}(aCompl)} I_{\phi^{-1}(\mathbf{a})}$$
16: $\quad\quad$ **end if**
17: $\quad$ **until** ($aCompl \neq \varepsilon \wedge$ U-ARGS($aCompl$) $\neq \emptyset$)
18: $\quad$ **if** $prefcand \neq \emptyset$ **then**
19: $\quad\quad E_p := E_p \cup \{\text{I-ARGS}(prefcand)\}$
20: $$cnf := cnf \wedge \neg \left(\bigwedge_{\mathbf{a} \in \text{I-ARGS}(prefcand)} I_{\phi^{-1}(\mathbf{a})} \wedge \bigwedge_{\mathbf{a} \in \text{O-ARGS}(prefcand)} O_{\phi^{-1}(\mathbf{a})} \wedge \bigwedge_{\mathbf{a} \in \text{U-ARGS}(prefcand)} U_{\phi^{-1}(\mathbf{a})} \right)$$
21: $\quad$ **end if**
22: **until** ($prefcand \neq \emptyset$)
23: **if** $E_p = \emptyset$ **then**
24: $\quad E_p = \{\emptyset\}$
25: **end if**
26: **return** E_p

fects both loops. The latter, *cnfdf*, keeps track of the search within the optimisation process, thus affecting the inner loop only.

At first, *cnf* is initialised (l. 4) to $\Pi_\Gamma \wedge \bigvee_{\mathbf{a} \in \mathcal{A}} I_{\phi^{-1}(\mathbf{a})}$: $\emptyset$ is excluded since it is always admissible. At l. 6 *cnfdf* is initialised to the value of *cnf* and, after entering the inner loop, the SAT solver is called over *cnfdf* returning a complete labelling *aCompl* (l. 9). This is a candidate to become a preferred labelling until either (i) a "bigger" complete labelling containing *aCompl* is found; or (ii) it is proven that there are no further complete labellings containing *aCompl*. To search for (i), at

line 13 each `in`-labelled (`out`-labelled) argument in *aCompl* is forced to be labelled `in` (`out`) until a preferred labelling is found, and to guide future searches towards a strictly bigger labelling at least one more argument is enforced to be labelled `in`.

The inner loop is exited when either (i) *SatS* returns no solution (ε); or (ii) I-ARGS(*aCompl*)$\cup$O-ARGS(*aCompl*) $= \mathcal{A}$. In the first case, the *prefcand* preferred labelling candidate found at the previous iteration cannot be extended to a complete labelling including a greater set of `in`-labelled arguments, thus it is a preferred labelling[i]. In the second case, this is due to the fact that there are no undecided arguments.

Once a preferred labelling is found, the set of preferred extensions is enriched (l. 19) with I-ARGS(*prefcand*) and $\neg$*prefcand* is added as a further constraint (line 20) within *cnf* before executing the external loop once again.

The implemented procedure for computing the grounded labelling is analogous, with the difference that the set of `undec`-labelled arguments—instead of the set of `in`-labelled arguments—is maximised in the inner loop.

As per checking the credulous acceptance of an argument $\mathbf{x}$ w.r.t. the:

- grounded semantics: jArgSemSAT checks whether $\mathbf{x}$ is in the set of `in`-labelled arguments of the grounded labelling;
- stable semantics: jArgSemSAT checks whether there is a solution to the formula $\Pi'_\Gamma \wedge I_{\phi^{-1}(\mathbf{x})}$;
- preferred semantics: jArgSemSAT checks whether there is a solution to $\Pi_\Gamma \wedge I_{\phi^{-1}(\mathbf{x})}$, which implies that there is a maximum labelling—i.e. a preferred labelling—containing that argument.

As per checking the skeptical acceptance of an argument $\mathbf{x}$ w.r.t. the:

- grounded semantics: since the grounded extension is unique, it is equivalent to check the credulous acceptance of $\mathbf{x}$;
- stable semantics:
  (1) if there is a solution to the formula $\Pi'_\Gamma \wedge O_{\phi^{-1}(\mathbf{x})}$ (equivalent to the question: does a stable labelling where $\mathbf{x}$ is `out` exist?) then jArgSemSAT returns False;
  (2) at this point, if there is a solution to the formula Π'_Γ (i.e. there exists at least a stable extension), then $\mathbf{x}$ belongs to the `in`-labelled sets for each stable labellings, and jArgSemSAT returns True (otherwise it returns False);
- preferred semantics: jArgSemSAT checks within Algorithm 1 whether $\mathbf{x}$ is not in the set of `in`-labelled arguments of a found preferred labelling and returns False in this case. jArgSemSAT returns True otherwise.

[i]In the case this happens at the first execution, it means that $\emptyset$ is the only preferred extension, cf. lines 23–24.

4. System design

jArgSemSAT is a mature application that now exists in four different versions:

(1) Stand-alone application: this guarantees compatibility with the Probo interface for the International Competition on Computational Models of Argumentation (ICCMA)[22];
(2) Dung-O-Matic (DoM)[11] compatible library: this ensures compatibility for works already using DoM such as CISpaces[6];
(3) Tweety compatible library[12]: we proudly support the Tweety project whose aim is to provide a general framework for implementing and testing knowledge representation formalisms;
(4) ArgTech [13] compatible web-service: we created a Tomcat web-service exporting jArgSemSAT with ArgTech-compatible RESTful interfaces.

jArgSemSAT is freely (MIT licence) available on SourceForge[j] and as Maven projects directly accessible from the central repository[k]. It is composed by two `jar` files and a `war` file.

`jArgSemSAT-VERSION.jar` provides both the stand-alone application compatible with the Probo interface and the DoM compatible library: we chose not to distribute the library without the Probo interface to facilitate future experiments also from different research groups and to improve the awareness in the community of the ICCMA competition.

Figure 1 depicts the UML graph of the main classes included in the `net.sf.jargsemsat.jarsemsat.alg`, namely those implementing the algorithms for computing complete, grounded, preferred, stable, and semi–stable[l] extensions. In particular, two methods are particularly important in `CompleteSemantics` class: `basicComplete` and `satlab`.

`basicComplete` computes—depending on the parameter `Encoding` passed to it—the propositional formula for complete labellings among all those introduced in Section 3.1. Instead, `satlab` is the method which deals with SAT solvers: it requires as input an object of type `SATFormulae`, an empty `Labelling` to store the result of the computation, and a `DungAF`.[m] It returns `true` if a satisfiable assignment is found, `false` otherwise.

`jArgSemSATTweety-VERSION.jar` is a self-contained, Tweety-compatible, library: it includes `jArgSemSAT-VERSION.jar` and provides a Tweety-compatible interface.

[j] `https://sourceforge.net/projects/jargsemsat/`
[k] `http://search.maven.org/`
[l] semi–stable semantics implementation is still experimental and, as such, not described in this document.
[m] `SATFormulae`, `Labelling`, and `DungAF` belong to the package `net.sf.jargsemsat.jargsemsat.datastructures`.

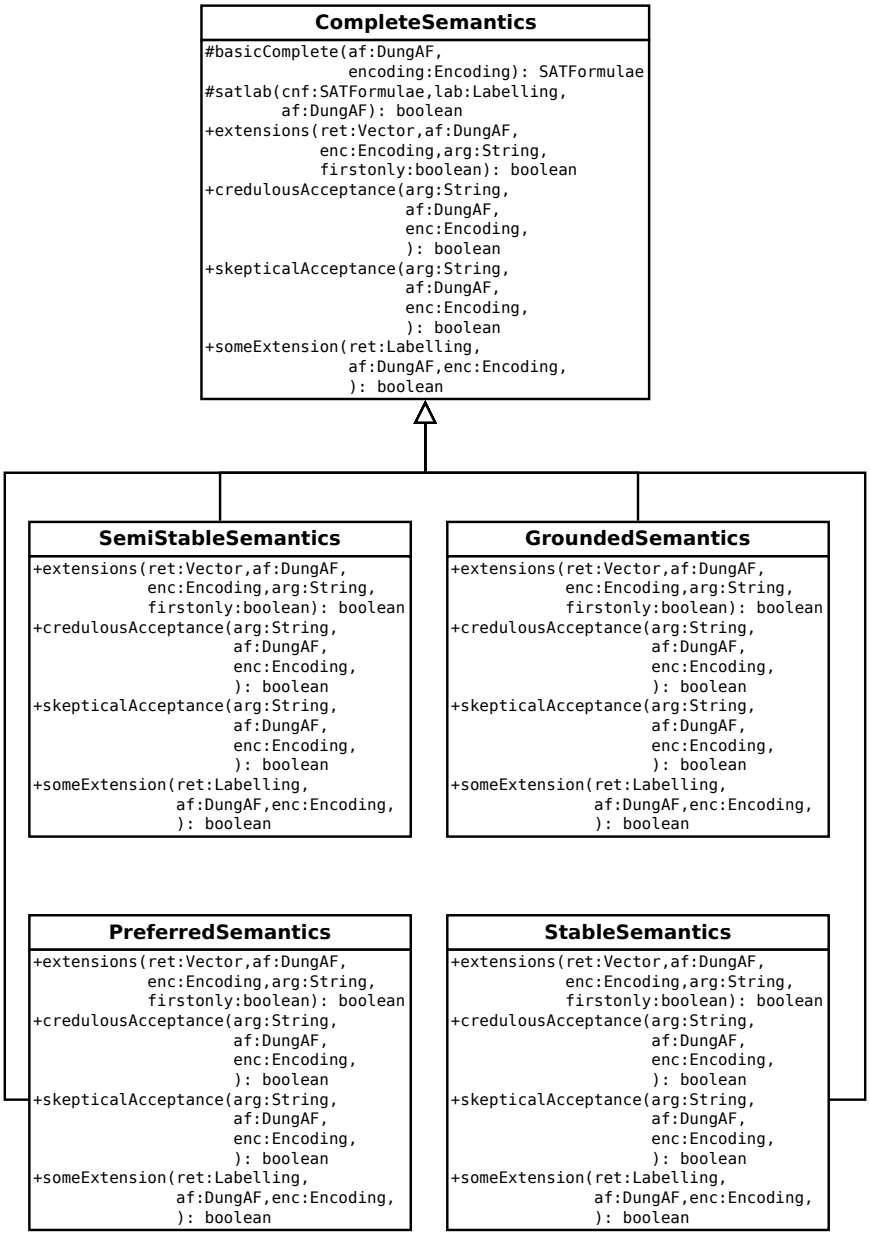

Fig. 1. UML diagram of the core components of jArgSemSAT

`jArgSemSATWeb-VERSION.war` is a self-contained Tomcat[n] web-service archive compatible with ArgTech[o] specifications. This web-service is also available free-of-

[n] `http://tomcat.apache.org/`
[o] `http://ws.arg.tech/`

charge—with best effort SLA—at `http://cicero.cs.cf.ac.uk/jArgSemSATWeb/restapi/argtech/` . Its source code is also freely available.

4.1. *Stand-alone application*

jArgSemSAT exports the same command line interface of ArgSemSAT, which is a superset of the Probo interface. In addition to the options discussed in previous work[22], jArgSemSAT allows the user to choose (1) the SAT solver to be used—Sat4j is the default; and (2) the encoding to use—111100, equivalent to $C^{\leftrightarrow}_{\mathtt{in}} \wedge C^{\leftrightarrow}_{\mathtt{out}}$, is the default.

4.2. *Dung-O-Matic (DoM) compatible library*

jArgSemSAT exports methods whose signature are compatible with Dung-O-Matic[11]: those methods encapsulate the code for calling jArgSemSAT with the default configurations, and on data-structures that reside on memory instead on a file.

Therefore, the following snippet code:

```
Vector<String> args
             = new Vector<String>();
args.add("a");
args.add("b");
Vector<String []> atts
             = new Vector<String []>();
atts.add(new String []{"a", "b"});
new DungAF(args, atts).getStableExts();
```

is valid if either DoM or jArgSemSAT library is imported.

4.3. *Tweety compatible library*

Tweety libraries[12] implement abstract argumentation reasoning procedures in the package `net.sf.tweety.arg.dung`. For instance, Figure 2 depicts a simple piece of code for creating a Dung's argumentation framework with two arguments, **a**, and **b**, where **a** attacks **b**, and for enumerating its preferred extensions using the Tweety libraries.

In order to guarantee the full compatibility with the Tweety libraries[12], and to reduce the burden on programmers already using them, `jArgSemSATTweety` extends the `net.sf.tweety.arg.dung.GroundReasoner`, `net.sf.tweety.arg.dung.PreferredReasoner`, and `net.sf.tweety.arg.dung.StableReasoner`, overriding only the method `computeExtensions` in each of them. Therefore, by importing `jArgSemSATTweety` and using `net.sf.jargsemsat.jArgSemSATTweety.PreferredReasoner` instead of `net.sf.tweety.arg.dung.PreferredReasoner`, the software will automatically

```
DungTheory at = new DungTheory();
at.add(new Argument("a"));
at.add(new Argument("b"));
at.add(new Attack(a,b));
PreferredReasoner r = new PreferredReasoner(at);
System.out.println(r.getExtensions());
```

Fig. 2. Creating a simple Dung's Argumentation Framework using Tweety libraries and enumerating its preferred extensions.

use Algorithm 1 for enumerating preferred extensions. As for the DoM compatible library, jArgSemSAT uses the default configurations only.

4.4. *ArgTech compatible web-service*

As presented in `http://ws.arg.tech/`, the ArgTech web-service solver[13] for abstract argumentation problems requires a POST message with the following fields:

- `arguments`, type String Array, e.g. `["A","B", "Arg_1"]`;
- `attacks`, type String Array, e.g. `["(A,B)", "(B,Arg_1)"]`;
- `semantics`, type String, one of `grounded`, `preferred`, `stable`, `semistable`[p].

For instance, the following JSon structure

```
{"arguments":["a","b"],
 "attacks":["(a,b)"],
 "semantics":"stable"}
```

is a valid POST request for `jArgSemSATWeb`. jArgSemSAT is then invoked with the default configurations only.

5. Evaluation

In this section, we present the result of a large experimental analysis comparing the performance of jArgSemSAT with respect to ArgSemSAT[23], Dung-o-Matic[11] and top ICCMA2015 solvers of tracks related to preferred and stable semantics: namely, ASPARTIX-D[24], Cegartix[25], CoQuiAAS[9], and LabSATSolver[26]. To complete the picture, we also include an analysis on complete extensions in Appendix A.

The aim of this section is to provide a good overview of the performance gap between the Java-based proposed system and the more efficient C++ implementations commonly exploited in competitions and academic studies. Moreover, the comparison with Dung-o-Matic (hereinafter DoM), helps to compare the performance of

[p]To ensure full compatibility, jArgSemSAT contains an experimental implementation of an algorithm for enumerating semi-stable extensions, see Section 6.

jArgSemSAT with a Java-based tool that is currently exploited in research-grade prototypes such as those presented in Refs. 13, 6. Among those, CISpaces[6] is now under analysis for transitioning into commercial products.

5.1. *Experimental Setup*

The experiments were performed on a cluster with computing nodes equipped with 2.4 Ghz Dual Core AMD Opteron™ processors, 4 GB of RAM and Linux operating system. A cutoff of 600 seconds (10 minutes) —as in ICCMA2015— was imposed for solving each problem on a single AF.

jArgSemSAT can exploit any SAT solver that supports the DIMACS format. In the current version, it comes with the Java-based SAT solver Sat4j[20] integrated. This guarantees the maximum portability of the proposed system, and minimises the overhead due to external system calls. Therefore, Sat4j is the SAT solver used by jArgSemSAT in this experimental analysis, unless differently specified. For the solvers selected according to their ICCMA2015 performance, the latest available version has been considered in this analysis[q]. DoM has been provided with a Probo compatible command line interface by reusing part of the code wrote for jArgSemSAT. For each solver we recorded the overall result: success, crashed, timed-out or ran out of memory.

Experiments have been conducted on the ICCMA2015 benchmark, which is a set of 192 randomly generated AFs. They have been generated considering three different graph models, in order to provide different levels of complexity. More details can be found on the ICCMA website. Here we considered credulous acceptance DC-σ, skeptical acceptance DS-σ and extensions enumeration EE-σ problems for $\sigma \in \{\text{stable}, \text{preferred}\}$, as they are the most computationally difficult problems among those included in the competition.

Performance are measured in terms of IPC score and Penalised Average Runtime. The IPC score, borrowed from the planning community[r], is defined as follows. For each AF, each system gets a score of $1/(1+\log_{10}(T/T^*))$, where T is its execution time and T^* the best execution time among the compared systems, or a score of 0 if it fails in that case. Runtimes below 1.0 sec get by default the maximal score of 1.

The Penalised Average Runtime (PAR score) is a real number which counts (i) runs that fail to solve the considered problem as ten times the cutoff time (PAR10) and (ii) runs that succeed as the actual runtime. PAR scores are commonly used in automated algorithm configuration, algorithm selection, and portfolio construction[28] because using them allows runtime to be considered while still placing a strong emphasis on high instance set coverage.

[q]Solvers have been retrieved in September 2015 from the corresponding websites, provided in Ref. 27.
[r]http://www.icaps-conference.org/index.php/Main/Competitions

5.2. *Comparison with the State of the Art of Abstract Argumentation Solvers*

Table 3. Performance achieved by jArgSemSAT, and the top three participants of the corresponding ICCMA2015 tracks on preferred semantics. Results are shown in terms of IPC score (maximum achievable is 192.0), PAR10 and percentages of success, and ordered according to PAR10. ICCMA15 ranking is also reported.

DC-PR				
Solver	**ICCMA15 Rank**	**IPC score**	**PAR10**	**% Success**
ArgSemSAT	1	164.7	3.2	100.0
LabSATSolver	2	167.9	3.4	100.0
jArgSemSAT		135.4	30.7	100.0
CoQuiAAS	3	186.4	63.6	98.9
DS-PR				
Solver	**ICCMA15 Rank**	**IPC score**	**PAR10**	**% Success**
ArgSemSAT	1	171.2	5.9	100.0
Cegartix	2	161.2	7.9	100.0
LabSATSolver	3	171.0	12.6	100.0
jArgSemSAT		136.9	40.4	100.0
EE-PR				
Solver	**ICCMA15 Rank**	**IPC score**	**PAR10**	**% Success**
Cegartix	1	157.8	15.2	100.0
ArgSemSAT	2	147.7	66.1	99.5
jArgSemSAT		122.7	194.2	97.9
CoQuiAAS	3	172.9	218.2	96.9

This set of experiments focuses on assessing the performance gap between the proposed jArgSemSAT, ArgSemSAT and the top three solvers of the ICCMA2015 stable and preferred semantics tracks.

Table 3 shows the results, in terms of IPC score, PAR10 and percentage of successfully analysed frameworks, of the performed comparison on the preferred semantics tracks. Results of stable semantics tracks are reported in Table 4.

According to results shown in Table 3 and Table 4, we can safely state that jArgSemSAT is an off-the-shelf and ready-to-use efficient solver for computationally complex abstract argumentation problems. In terms of AFs successfully analysed, jArgSemSAT shows performance that are very similar to ArgSemSAT and to the winner of the considered tracks, though it is slower according to PAR10 and IPC score. Tables 3 and 4 also allow one to identify the performance gain given by the C++ implementation. Admittedly, ArgSemSAT is faster than jArgSemSAT; however,

the performance gap is not critical.

Table 4. Performance achieved by jArgSemSAT, ArgSemSAT, and the top three participants of the corresponding ICCMA2015 tracks on stable semantics. Results are shown in terms of IPC score (maximum achievable is 192.0), PAR10 and percentages of success, and ordered according to PAR10. ICCMA15 ranking is also reported. The provided ordering and the ICCMA15 ranking differ for DC-ST, due to close performance and a slightly different hardware configuration; and for DS-ST due to a bugfix.

DC-ST				
Solver	**ICCMA15 Rank**	**IPC score**	**PAR10**	**% Success**
ASPARTIX-D	1	183.2	1.7	100.0
LabSATSolver	3	186.7	1.7	100.0
ArgSemSAT	2	172.5	2.7	100.0
jArgSemSAT		138.1	30.3	100.0
DS-ST				
Solver	**ICCMA15 Rank**	**IPC score**	**PAR10**	**% Success**
ASPARTIX-D	1	173.3	2.6	100.0
ArgSemSAT	7	150.8	11.2	100.0
LabSATSolver	2	138.0	18.6	100.0
jArgSemSAT		125.1	42.0	100.0
CoQuiAAS	3	180.2	65.7	99.0
EE-ST				
Solver	**ICCMA15 Rank**	**IPC score**	**PAR10**	**% Success**
ASPARTIX-D	1	172.6	5.4	100.0
ArgSemSAT	2	144.7	51.3	99.5
jArgSemSAT		122.9	82.9	99.5
CoQuiAAS	3	184.1	135.0	97.9

According to the results shown in Tables 3 and 4, jArgSemSAT is comparable with the state of the art of solvers for abstract argumentation problems. Only in the DC-ST and DS-PR tracks jArgSemSAT is not among the best three considered solvers.

5.3. *Comparison with the State of the Art of Off-the-Shelf Solvers*

This analysis aims at comparing jArgSemSAT with the only available Java-based, off-the-shelf solver DoM.

Compared to the existing off-the-shelf implementation, DoM, jArgSemSAT sta-

Table 5. Performance achieved by jArgSemSAT and DoM on the corresponding ICCMA2015 tracks. Results are shown in terms of IPC score (maximum achievable is 192.0), PAR10 and percentages of success. "–" indicates that the solver does not support the considered problem for the given semantic.

Track	**IPC score (192.0)**		**PAR10**		**% Success**	
	jArg	DoM	jArg	DoM	jArg	DoM
DC-PR	192.0	–	30.7	–	100.0	–
DS-PR	192.0	–	40.4	–	100.0	–
EE-PR	187.7	31.5	194.2	4332.1	97.9	28.1
Track	**IPC score (192.0)**		**PAR10**		**% Success**	
	jArg	DoM	jArg	DoM	jArg	DoM
DC-ST	192.0	–	30.3	–	100.0	–
DS-ST	192.0	–	41.9	–	100.0	–
EE-ST	190.0	31.6	82.9	4331.6	99.5	28.1

tistically significantly outperforms[s] DoM in both considered EE-σ problems in terms of runtime (WSRT, $p < 0.05$), cf. Table 5. It is also noticeable the fact that DoM is able to successfully analyse a small number of benchmark AFs. Interestingly, we observed that DoM is the only system—among considered—that does not show a statistically significant difference (WSRT, $p = 0.90$) in the CPU-time required for enumerating stable and preferred extensions of a given AF. Moreover, it is worth noting that DoM demonstrated to be very sensitive to the structure of the AFs to solve. Specifically, it did not solve—with respect to the considered enumeration problems—any of the graphs generated by using the "GroundedGenerator". Such graphs are characterised by a very large grounded extension and a large number of nodes.

5.4. *Importance of the SAT Solver*

This analysis investigates the impact of different SAT solvers on the performance of jArgSemSAT. Specifically, we considered the Java-based SAT solver Sat4j—which guarantees high portability and easy usage— and glucose3.0[19] that is written in C++. It should be noted that Sat4j can keep the learned constraints between two satisfiability checks, in order to exploit the gained knowledge in subsequent calls on very similar CNFs. For the sake of modularity, and for providing a more objective comparison, this feature is not exploited in the jArgSemSAT framework.

We are aware that the exploitation of a C++ software can pose some strong portability issues, mainly due to compilers and libraries, but C++ solvers are gen-

[s]In the following we rely on the Wilcoxon Signed-Rank Test (WSRT) as a paired difference test to establish statistically significant difference[29].

Table 6. Performance achieved by jArgSemSAT exploiting either Sat4j or glucose3.0 on the ICCMA2015 benchmark. For the sake of comparison, also the performance of ArgSemSAT are provided. Results are shown in terms of IPC score (maximum achievable is 192.0), percentages of success, percentages of AFs in which the system has been the fastest and PAR10. Values in bold indicate the best results.

	jArgSemSAT		ArgSemSAT
	Sat4j	glucose3.0	
IPC score	**151.2**	146.0	184.4
% success	97.9	**99.5**	99.5
% best	**13.5**	9.9	65.6
PAR10	194.2	**81.7**	66.1

erally believed to be faster than corresponding Java-based systems. Therefore, here we are interested in measuring such performance gap, in order to make jArgSemSAT users aware of the importance of the solver. However, it should be noted that Sat4j has been included in the overall jArgSemSAT framework, while glucose3.0 has to be executed through PIPE communication system among processes.

For stressing the importance of SAT solvers, thus obtaining a better understanding of their impact on jArgSemSAT performance, we considered the empirically most computationally expensive tasks. According to the results shown in Table 3 and Table 4, the problem of enumerating the preferred extensions (EE-PR) of a given AF requires the largest amount of CPU-time. This can be easily derived by the PAR10 scores and the percentage of successfully analysed AFs by the considered solvers.

Table 6 shows the results of the comparison between jArgSemSAT exploiting the Java-based Sat4j solver and jArgSemSAT using the C++ glucose3.0 SAT solver. For the sake of comparison, also the performance of the ArgSemSAT system are shown. Interestingly, results shown in Table 6 seem to indicate that the use of glucose3.0 does not provide a remarkable performance improvement. In particular, the exploitation of the external C++ solver has a detrimental effect on the performance of jArgSemSAT in terms of IPC score and number of times the approach has been the fastest. However, when a closer look to the observed performance is taken, an interesting pattern emerges. Surprisingly, the performance of considered SAT solvers are not directly related to the number of preferred extensions, i.e. there is no direct relation between the number of times the solver is called by jArgSemSAT and the runtime. Furthermore, Sat4j improves the performance of jArgSemSAT on AFs that can be solved in less than—approximately—50 CPU-time seconds; on more complex AFs, the use of glucose3.0 is usually beneficial. Mainly because of that, the Wilcoxon test indicates that the performance of the compared systems are significantly different (WSRT, $p = 0.01$). The ability of glucose3.0 to handle empirically complex AFs, is confirmed by the fact that the use of glucose3.0 allows jArgSemSAT

to solve, within the given time, a few more AFs from the considered ICCMA2015 benchmark.

6. Conclusion

In this paper we present jArgSemSAT, an efficient off-the-shelf solver for abstract argumentation problems. In the previous sections we give evidence of how jArgSemSAT not only is compatible with the current off-the-shelf solver, namely Dung-O-Matic[11], and with the Tweety libraries[12]; not only exists in a web-service version compatible with ArgTech technologies[13]—and we made it freely available at `http://cicero.cs.cf.ac.uk/jArgSemSATWeb/restapi/argtech/`—; but it is among the best solvers in particular for most of the tracks of the ICCMA2015 competition associated to the two semantics with highest computational complexity, namely stable and preferred semantics. As discussed in Section 5.4, the choice of the oracle can significantly impact the performance of a solver. This is true not only for the proposed jArgSemSAT solver, but also for competitors. Indeed, as noticed by one of the reviewers, it would be interesting how other systems such as CoQuiAAS would perform with a more powerful MSS enumerator, such as the one proposed in Ref. 30.

Currently, jArgSemSAT is used within CISpaces[6] that has been our main use-case. CISpaces (Collaborative Intelligence Spaces) is a tool mostly written in Java—only the GUI is written in Python—to help analysts in acquiring, evaluating and interpreting information. Indeed, the aim of intelligence analysis is to make sense of information that is often conflicting or incomplete, and to weigh competing hypotheses that may explain a situation. This imposes a high cognitive load on analysts, and there are few automated tools to aid them in their task. CISpaces assists analysts in reasoning with different types of evidence: analysts are supported in structuring evidence using argumentation schemes, and in identifying plausible hypotheses via the computation of preferred extensions.

By adopting jArgSemSAT, CISpaces now computes the preferred extensions of average analysis almost instantaneously: before, using Dung-O-Matic, it required 60 seconds or more. This was becoming a serious impediment to the adoption of CISpaces for training new analysts—its main goal—and it was listed as one of the improvements needed to be addressed before moving the project towards a commercial transition. After the excellent performance of ArgSemSAT at ICCMA2015, we decided to re-code it in Java to ease the integration. This also satisfied the other requirement to make it available as a replacement for Dung-O-Matic and integrate it within the Tweety libraries—both written in Java. Finally, it also greatly simplified the task of producing a web-service interface.

Moreover, jArgSemSAT allows CISpaces to use its probabilistic argumentation engine in real analysis. Indeed, CIspaces includes a probabilistic argumentation engine[31,32] that heavily resides on preferred extensions computed on probabilistic manipulation of AFs. Therefore, the preferred extension enumeration solver needs

to be invoked an exponential number of times.

As discussed in Section 3.3 of Li's work[32], when an argumentation framework is analysed from a probabilistic standpoint, it is necessary to know the semantics extensions of all the possible combinations of sub-graphs. Therefore, for a simple argumentation framework with two arguments **a**, and **b**, this requires to enumerate the semantics extensions of the frameworks: $\langle\emptyset,\emptyset\rangle$, $\langle\{\mathbf{a}\},\emptyset\rangle$, $\langle\{\mathbf{b}\},\emptyset\rangle$, $\langle\{\mathbf{a},\mathbf{b}\},\emptyset\rangle$, $\langle\{\mathbf{a},\mathbf{b}\},\{\langle\mathbf{a},\mathbf{b}\rangle\}\rangle$, $\langle\{\mathbf{a},\mathbf{b}\},\{\langle\mathbf{b},\mathbf{a}\rangle\}\rangle$, $\langle\{\mathbf{a},\mathbf{b}\},\{\langle\mathbf{a},\mathbf{b}\rangle,\langle\mathbf{b},\mathbf{a}\rangle\}\rangle$, $\langle\{\mathbf{a}\},\{\langle\mathbf{a},\mathbf{a}\rangle\}\rangle$, $\langle\{\mathbf{a},\mathbf{b}\},\{\langle\mathbf{a},\mathbf{a}\rangle\}\rangle$, $\langle\{\mathbf{a},\mathbf{b}\},\{\langle\mathbf{a},\mathbf{a}\rangle,\langle\mathbf{a},\mathbf{b}\rangle\}\rangle$, $\langle\{\mathbf{a},\mathbf{b}\},\{\langle\mathbf{a},\mathbf{a}\rangle,\langle\mathbf{b},\mathbf{a}\rangle\}\rangle$, $\langle\{\mathbf{a},\mathbf{b}\},\{\langle\mathbf{a},\mathbf{a}\rangle,\langle\mathbf{a},\mathbf{b}\rangle,\langle\mathbf{b},\mathbf{a}\rangle\}\rangle$, $\langle\{\mathbf{b}\},\{\langle\mathbf{b},\mathbf{b}\rangle\}\rangle$, $\langle\{\mathbf{a},\mathbf{b}\},\{\langle\mathbf{b},\mathbf{b}\rangle\}\rangle$, $\langle\{\mathbf{a},\mathbf{b}\},\{\langle\mathbf{b},\mathbf{b}\rangle,\langle\mathbf{a},\mathbf{b}\rangle\}\rangle$, $\langle\{\mathbf{a},\mathbf{b}\},\{\langle\mathbf{b},\mathbf{b}\rangle,\langle\mathbf{b},\mathbf{a}\rangle\}\rangle$, $\langle\{\mathbf{a},\mathbf{b}\},\{\langle\mathbf{b},\mathbf{b}\rangle,\langle\mathbf{a},\mathbf{b}\rangle,\langle\mathbf{b},\mathbf{a}\rangle\}\rangle$, $\langle\{\mathbf{a},\mathbf{b}\},\{\langle\mathbf{a},\mathbf{a}\rangle,\langle\mathbf{b},\mathbf{b}\rangle\}\rangle$, $\langle\{\mathbf{a},\mathbf{b}\},\{\langle\mathbf{a},\mathbf{a}\rangle,\langle\mathbf{b},\mathbf{b}\rangle,\langle\mathbf{a},\mathbf{b}\rangle\}\rangle$, $\langle\{\mathbf{a},\mathbf{b}\},\{\langle\mathbf{a},\mathbf{a}\rangle,\langle\mathbf{b},\mathbf{b}\rangle,\langle\mathbf{b},\mathbf{a}\rangle\}\rangle$, $\langle\{\mathbf{a},\mathbf{b}\},\{\langle\mathbf{a},\mathbf{a}\rangle,\langle\mathbf{b},\mathbf{b}\rangle,\langle\mathbf{a},\mathbf{b}\rangle,\langle\mathbf{b},\mathbf{a}\rangle\}\rangle$.

In order to compute the results for those probabilistic approaches, semantics extensions must be computed exhaustively for all the possible sub-graphs. While there is some work in the dynamics in abstract argumentation (e.g. Ref. 33) to produce efficient algorithms for reusing partially computed results, possibly exploiting the concept of Input/Output multipoles[34], efficient algorithms for computing such results are surely needed.

In the case of CISpaces, while Dung-O-Matic limited the use of the probabilistic argumentation engine to toy examples of less than ten arguments, and still requiring between 30-90 seconds, jArgSemSAT makes it available for real analysis involving up to 50/60 arguments with solutions within 10 seconds.

Therefore, jArgSemSAT has positively contributed to push the research grade prototype CISpaces towards a plan for transitioning into a commercial product. Indeed, jArgSemSAT helped CISpaces to receive positive qualitative feedback from trained analysts chosen to evaluate it.

The future of jArgSemSAT, in our view, lays in supporting all the research community to build and exploit argumentation-based tools. That is the reason that motivated us in providing a free-of-charge, but clearly with best-effort only SLA, web-service interface to jArgSemSAT. From a technical perspective, we need to ultimate the technical documentation and we plan to include support for the remaining semantics, notably semi-stable[2]—that is already supported in an experimental, non-optimised version.

We will also create a web-based interface for goal-driven manipulation and evaluation of *AF*s. Currently, most of the web-based interfaces to argumentation tools, e.g. OVA[t], TOAST[u], Aspartix[v], Conarg[w], allow a user first to prepare an argumentation knowledge base, and then to run a solver on it. However, they do not provide "versioning" support: a user needs to manually keep track of the correspondence

[t] `http://ova.arg-tech.org/`
[u] `http://toast.arg-tech.org/`
[v] `http://rull.dbai.tuwien.ac.at:8080/ASPARTIX/index.faces`
[w] `http://www.dmi.unipg.it/conarg/`

between its action on the knowledge base and the effects on the computed extensions. Moreover, they do not provide "strategical" support: a user aiming at having a specific argument accepted has no guidance on how to achieve such a goal. The tool we plan to build will prove itself very useful for researchers on dynamics and argumentation[35,36,34] and more broadly for the entire argumentation community.

Acknowledgments

The authors thank (in alphabetical order) Prof. Chris Reed and Dr. Mark Snaith—University of Dundee—for their support in ensuring compatibility between jArgSemSAT and existing technologies at ArgTech. Moreover, we thank Dr. Matthias Thimm—Universität Koblenz-Landau—for his support in integrating jArgSemSAT with Tweety, and Dr. Alice Toniolo—University of Aberdeen—for integrating jArgSemSAT within CISpaces.

The authors would like to acknowledge the use of the University of Huddersfield Queensgate Grid in carrying out this work.

The authors also thank the anonymous reviewers for their helpful comments.

References

1. P. M. Dung, On the Acceptability of Arguments and Its Fundamental Role in Nonmonotonic Reasoning, Logic Programming, and n-Person Games, *Artificial Intelligence* **77**(2) (1995) 321–357.
2. M. Caminada, Semi-Stable Semantics, in *Proceedings of the 1st International Conference on Computational Models of Arguments (COMMA 2006)* (Liverpool, UK, 2006), pp. 121–130.
3. P. M. Dung, P. Mancarella and F. Toni, A dialectic procedure for sceptical, assumption-based argumentation, in *Prooceedings of the 1st International Conference on Computational Models of Arguments (COMMA 2006)*2006, pp. 145–156.
4. P. Baroni, M. Giacomin and G. Guida, SCC-recursiveness: a general schema for argumentation semantics, *Artificial Intelligence* **168**(1-2) (2005) 165–210.
5. N. Tamani, P. Mosse, M. Croitoru, P. Buche, V. Guillard, C. Guillaume and N. Gontard, An argumentation system for eco-efficient packaging material selection, *Computers and Electronics in Agriculture* **113** (apr 2015) 174–192.
6. A. Toniolo, T. J. Norman, A. Etuk, F. Cerutti, R. W. Ouyang, M. Srivastava, N. Oren, T. Dropps, J. A. Allen and P. Sullivan, Agent Support to Reasoning with Different Types of Evidence in Intelligence Analysis, in *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2015)*2015, pp. 781–789.
7. P. Baroni, M. Caminada and M. Giacomin, An introduction to argumentation semantics, *Knowledge Engineering Review* **26**(4) (2011) 365–410.
8. P. E. Dunne and M. Wooldridge, Complexity of abstract argumentation, in *Argumentation in AI* (Springer-Verlag, 2009) pp. 85–104.
9. J. Lagniez, E. Lonca and J. Mailly, Coquiaas: A constraint-based quick abstract argumentation solver, in *27th IEEE International Conference on Tools with Artificial Intelligence, (ICTAI 2015)*2015, pp. 928–935.
10. M. Thimm, S. Villata, F. Cerutti, N. Oren, H. Strass and M. Vallati, Summary report

of the first international competition on computational models of argumentation, *AI Magazine* (2015).
11. M. Snaith, J. Devereux, J. Lawrence and C. Reed, Pipelining argumentation technologies., in *Proceedings of the 3rd International Conference on Computational Models of Arguments (COMMA 2010)*2010, pp. 447–453.
12. M. Thimm, Tweety - a comprehensive collection of java libraries for logical aspects of artificial intelligence and knowledge representation, in *Proceedings of the 14th International Conference on Principles of Knowledge Representation and Reasoning (KR'14)*July 2014.
13. F. Bex, J. Lawrence, M. Snaith and C. Reed, Implementing the argument web, *Communications of the ACM* **56** (oct 2013) p. 66.
14. F. Cerutti, M. Vallati and M. Giacomin, jargsemsat: An efficient off-the-shelf solver for abstract argumentation frameworks, in *Proceedings of The 15th International Conference on Principles of Knowledge Representation and Reasoning (KR)* (AAAI, 2016).
15. P. Baroni, F. Cerutti, P. E. Dunne and M. Giacomin, Automata for Infinite Argumentation Structures, *Artificial Intelligence* **203** (may 2013) 104–150.
16. M. Caminada, On the Issue of Reinstatement in Argumentation, in *Proceedings of the 10th European Conference on Logics in Artificial Intelligence (JELIA 2006)*2006, pp. 111–123.
17. F. Cerutti, P. E. Dunne, M. Giacomin and M. Vallati, Computing Preferred Extensions in Abstract Argumentation: A SAT-Based Approach, in *TAFA 2013 Lecture Notes in Computer Science* **8306** (Springer-Verlag Berlin Heidelberg, 2014) pp. 176–193.
18. M. Caminada and D. M. Gabbay, A Logical Account of Formal Argumentation, *Studia Logica (Special issue: new ideas in argumentation theory)* **93**(2–3) (2009) 109–145.
19. G. Audemard and L. Simon, Lazy clause exchange policy for parallel sat solvers, in *Theory and Applications of Satisfiability Testing–SAT 2014* 2014 pp. 197–205.
20. D. Le Berre and A. Parrain, The Sat4j library, release 2.2 system description, *Journal on Satisfiability, Boolean Modeling and Computation* **7**(2010) (2010) 59–64.
21. N. Eén and N. Sörensson, An extensible sat-solver, in *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers*2003, pp. 502–518.
22. F. Cerutti, N. Oren, H. Strass, M. Thimm and M. Vallati, A Benchmark Framework for a Computational Argumentation Competition, in *Proceedings of the 5th International Conference on Computational Models of Argument (COMMA 2014)*2014, pp. 459–460.
23. F. Cerutti, M. Giacomin and M. Vallati, ArgSemSAT: Solving Argumentation Problems Using SAT, in *Proceedings of the 5th International Conference on Computational Models of Argument (COMMA 2014)*2014, pp. 455–456.
24. S. A. Gaggl and N. Manthey, Aspartix-d: Asp argumentation reasoning tool-dresden, in *System Descriptions of the First International Competition on Computational Models of Argumentation (ICCMA15)*2015.
25. W. Dvorák, M. Järvisalo, J. P. Wallner and S. Woltran, Cegartix: A sat-based argumentation system, in *Pragmatics of SAT Workshop (POS)*2012.
26. C. Beierle, F. Brons and N. Potyka, A software system using a sat solver for reasoning under complete, stable, preferred, and grounded argumentation semantics, in *KI 2015: Advances in Artificial Intelligence: 38th Annual German Conference on AI*, eds. S. Hölldobler, M. Krötzsch, R. Peñaloza and S. Rudolph2015, pp. 241–248.
27. M. Thimm and S. Villata, System Descriptions of the First International Competition on Computational Models of Argumentation (ICCMA'15), *CoRR* **abs/1510.05373** (2015).
28. H. H. Hoos, Automated algorithm configuration and parameter tuning, in *Autonomous*

search (Springer, 2012) pp. 37–71.
29. F. Wilcoxon, Individual comparisons by ranking methods, *Biometrics Bulletin* **1**(6) (1945) 80–83.
30. C. Mencía, A. Previti and J. Marques-Silva, Literal-based MCS extraction, in *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*2015, pp. 1973–1979.
31. H. Li, N. Oren and T. Norman, Probabilistic Argumentation Frameworks, in *Theorie and Applications of Formal Argumentation Lecture Notes in Computer Science* **7132** (Springer Berlin Heidelberg, 2012) pp. 1–16.
32. H. Li, *Probabilistic Argumentation*, PhD thesis, U. Aberdeen2015.
33. C. Cayrol, F. D. de Saint-Cyr and M.-C. Lagasquie-Schiex, Change in abstract argumentation frameworks: Adding an argument, *J. Artif. Int. Res.* **38** (May 2010) 49–84.
34. P. Baroni, G. Boella, F. Cerutti, M. Giacomin, L. van der Torre and S. Villata, On the Input/Output behavior of argumentation frameworks, *Artificial Intelligence* **217**(0) (2014) 144—-197.
35. B. Liao, L. Jin and R. C. Koons, Dynamics of argumentation systems: A division-based method, *Artificial Intelligence* **175** (jul 2011) 1790–1814.
36. R. Baumann, Splitting an Argumentation Framework, in *Proceedings of the 11th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2011)*2011, pp. 40–53.

Appendix A. Experimental Evaluation for Complete Extensions

Table 7. Performance achieved by jArgSemSAT, ArgSemSAT, and the top three participants of the corresponding ICCMA2015 tracks on complete extensions. Results are shown in terms of IPC score (maximum achievable is 192.0), PAR10 and percentages of success, and ordered according to PAR10. ICCMA15 ranking is also reported.

DC-CO				
Solver	**ICCMA15 Rank**	**IPC score**	**PAR10**	**% Success**
ArgSemSAT	1	179.0	2.5	100.0
LabSATSolver	3	182.0	2.8	100.0
ASPARTIX-D	2	171.2	3.1	100.0
jArgSemSAT		145.6	15.5	100.0
DS-CO				
Solver	**ICCMA15 Rank**	**IPC score**	**PAR10**	**% Success**
LabSATSolver	2	179.8	1.1	100.0
ASGL	1	191.5	2.3	100.0
ConArg	3	169.1	3.9	100.0
ArgSemSAT	4	161.0	4.5	100.0
jArgSemSAT		148.6	14.5	100.0
EE-CO				
Solver	**ICCMA15 Rank**	**IPC score**	**PAR10**	**% Success**
ASPARTIX-D	1	175.8	8.0	100.0
CoQuiAAS	3	181.3	46.1	99.4
jArgSemSAT		114.6	215.7	97.4
ArgSemSAT	2	131.2	254.0	96.4

Table 8. Performance achieved by jArgSemSAT and DoM on the corresponding ICCMA2015 tracks. Results are shown in terms of IPC score (maximum achievable is 192.0), PAR10 and percentages of success. "–" indicates that the solver does not support the considered problem for the given semantic.

Track	**IPC score (192.0)**		**PAR10**		**% Success**	
	jArg	DoM	jArg	DoM	jArg	DoM
DC-CO	192.0	–	15.5	–	100.0	–
DS-CO	192.0	–	14.5	–	100.0	–
EE-CO	180.1	39.3	215.7	4325.0	97.4	28.1